Machine Learning: (Bayesian) Neural Networks & Automatic Relevance Determination

Matthias Blaschke
University of Augsburg
Memorial University of Newfoundland

Gartner Hype Cycle for Artificial Intelligence, 2019

gartner.com/SmarterWithGartner

Source: Gartner

Gartner

Talk Outline

- Neural Networks

  (fancy)
- Bayesian Neural Networks

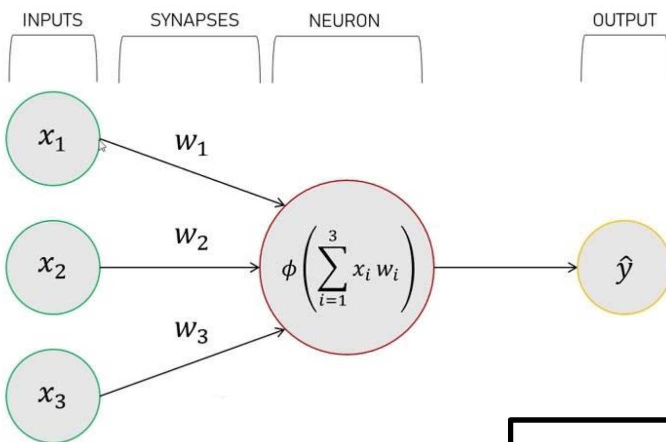
  (that's even fancier)
- Hierarchical Bayesian Neural Networks

  (Ok. That's awesome)
- Automatic Relevance Determination

  (concentrate on the important things…)

What is a Neural Network?

- Highly nonlinear function of the inputs

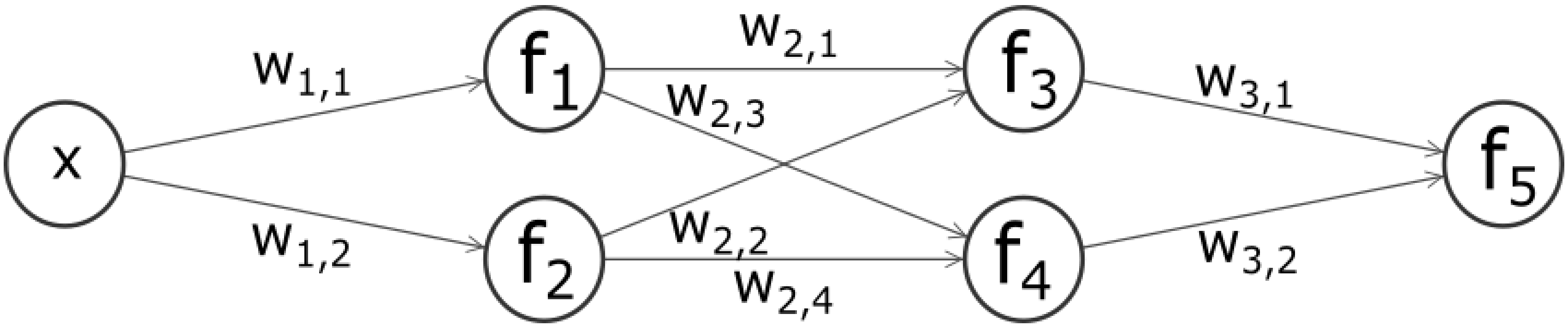


Activation function (nonlinearity)

$$\hat{y} = \phi\left(\sum_{i=1}^{3} x_i w_i\right) = \phi\left(\underbrace{x_1 w_1 + x_2 w_2 + x_3 w_3}_{\text{Activation } a}\right)$$

https://i.ytimg.com/vi/kft1AJ9WVDk/maxresdefault.jpg

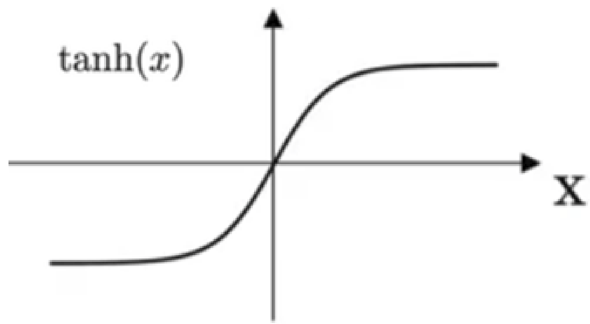
Deep Networks



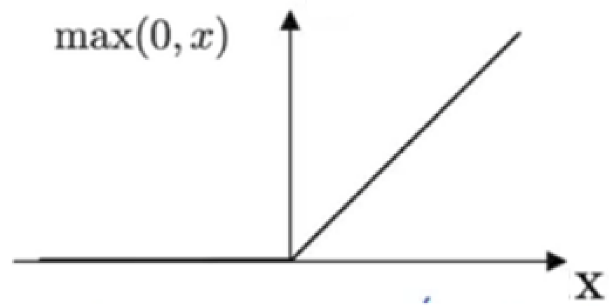
$$y = f_5\Bigg(w_{3,1} f_3\big(w_{2,1} f_1(w_{1,1}x)+w_{2,2} f_2(w_{1,2}x)\big)+\big(w_{3,2} f_4\big(w_{2,3} f_1(w_{1,1}x)+w_{2,4} f_2(w_{1,2}x)\big)\Bigg)$$

Common Activation Functions

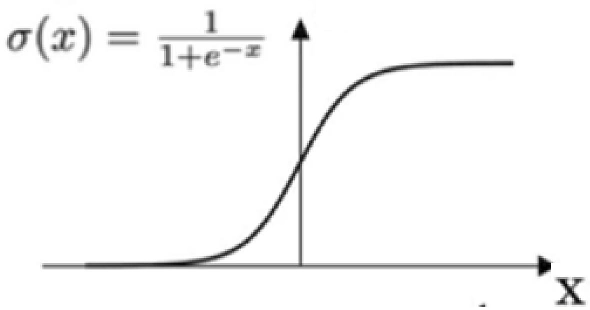
Hyper Tangent Function



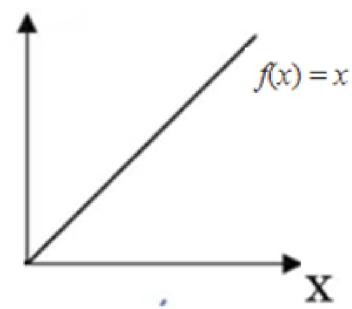
ReLU Function



Sigmoid Function



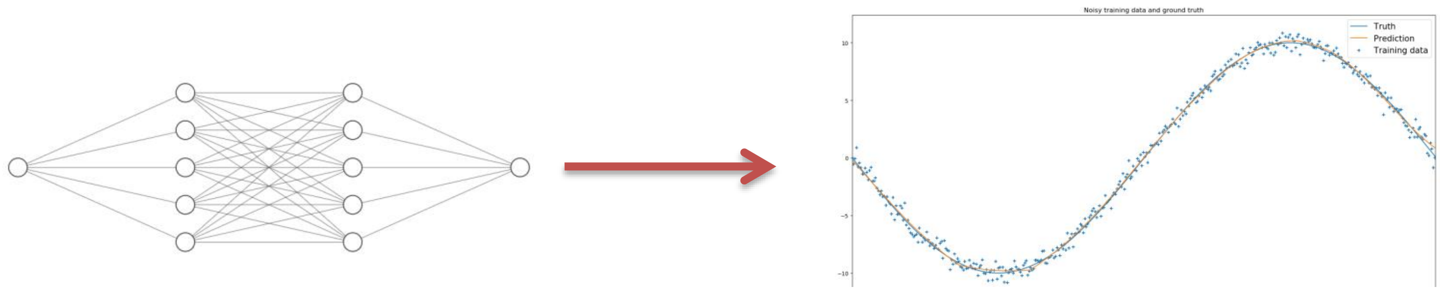
Identity Function



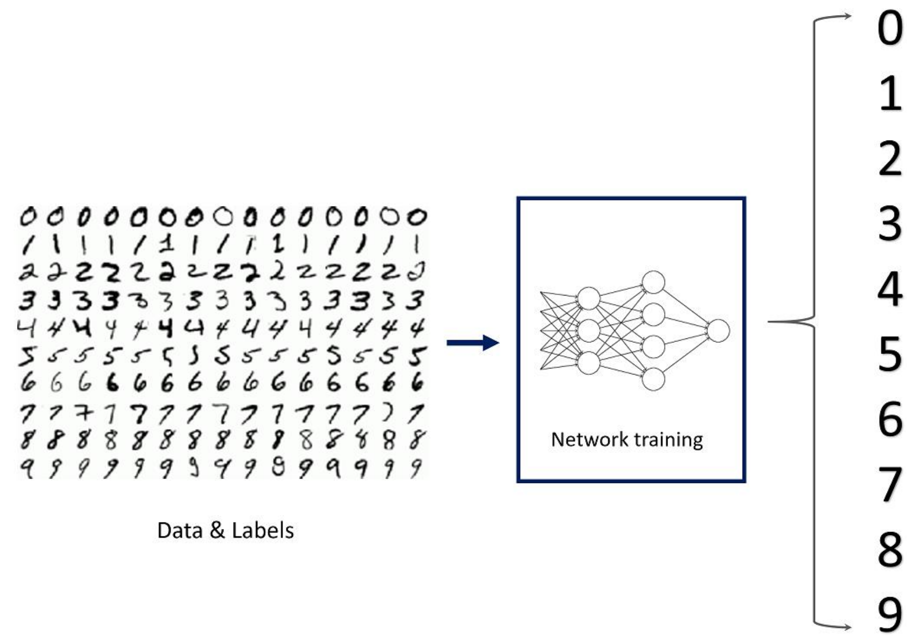
https://res.cloudinary.com/dyd911kmh/image/upload/f_auto,q_auto:best/v1547672259/4_jouacz.png

How can we use it ?

- Nonlinear regression



- Classification



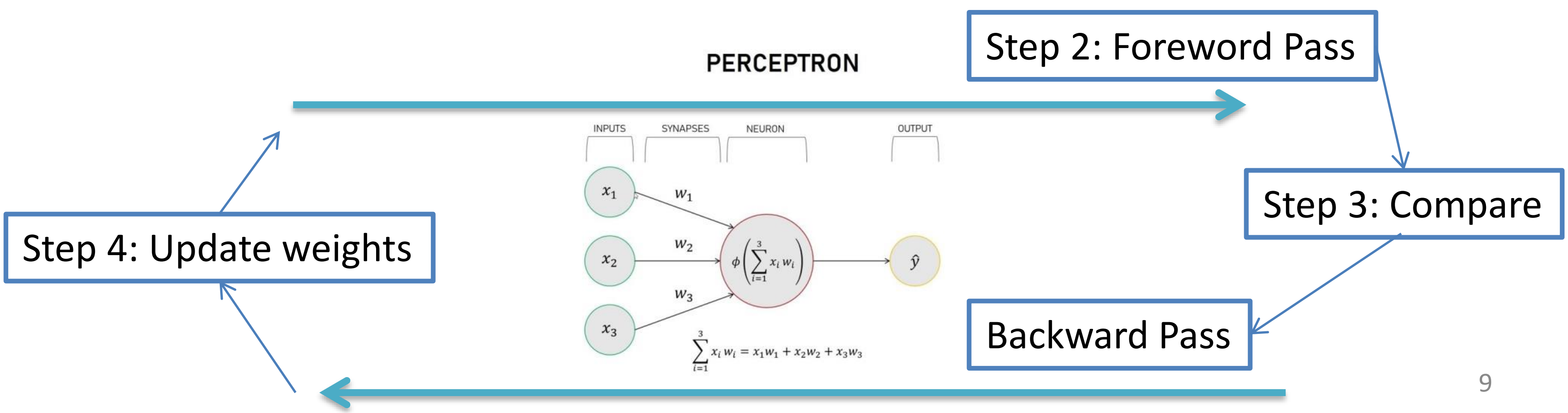
Universal approximation theorem

- Many different formulations. For example:

$\boxed{\text{Networks with Relu activation functions with width n+1 are sufficient to approximate any continuous convex function of n input variables}}$

How can we train it?

- **Step 1:** Initialize all weights randomly
- **Step 2:** Calculate output y with training data
- **Step 3:** Compare output y and desired output y
  →Define Loss function
- **Step 4:** Update weights



How do we update the weights?

- Backpropagation:

 -Error function for N outputs and Input j

$$E_j(w) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2$$

 -Total error function:

$$E(w) = \sum_{j=1}^{M} E_j(w)$$

<u>Batch:</u> M points from the training data

<u>Epoch:</u> The whole dataset was trained once

How do we update the weights?

- Update the weights using gradient descent:

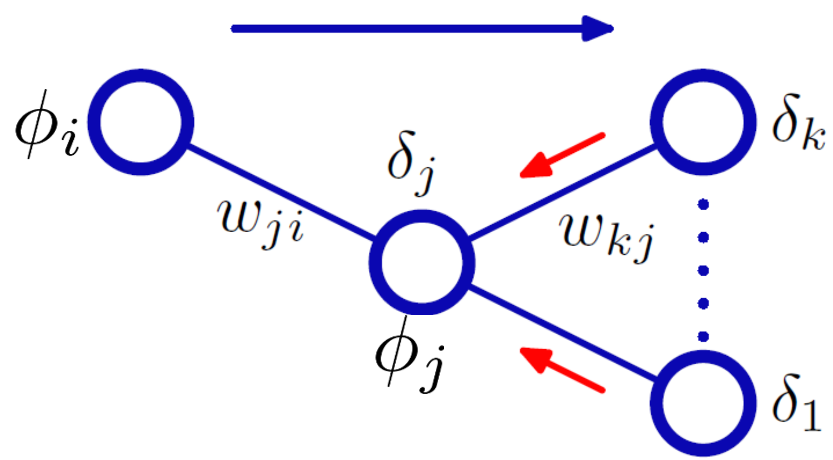
$$w^{(t+1)} = w^{(t)} - \eta \nabla E_n(w^{(t)}); \quad \eta = \text{learning rate}$$

- Calculate the gradient using backpropagation:

$$\frac{\partial E_n}{\partial w_{ji}} = \underbrace{\frac{\partial E_n}{\partial a_j}}_{\delta_j} \underbrace{\frac{\partial a_j}{\partial w_{ji}}}_{\phi_i = \phi(a_i)} = \delta_j \phi_i$$

How do we update the weights?

- Calculate δ_j
  - For output units $\delta_k = y_k - \hat{y}_k$
  - For hidden units



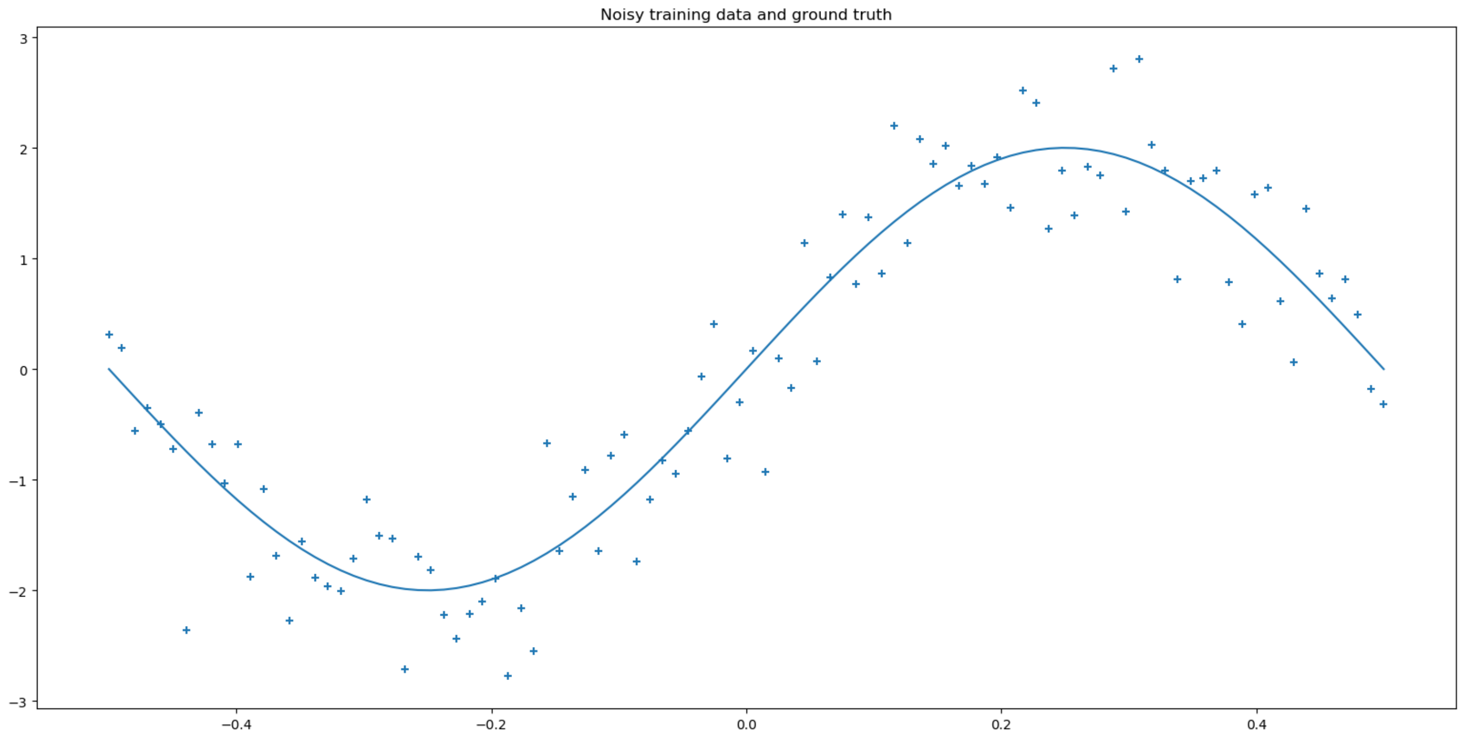
$$\delta_j = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j}$$

$$\delta_j = \phi_j' \sum_k w_{kj} \delta_k$$

BACKpropagation...

Regression example

- That's enough math. Let's have a look at an example:



Regression example

- Very easy to implement (using Python and Keras):

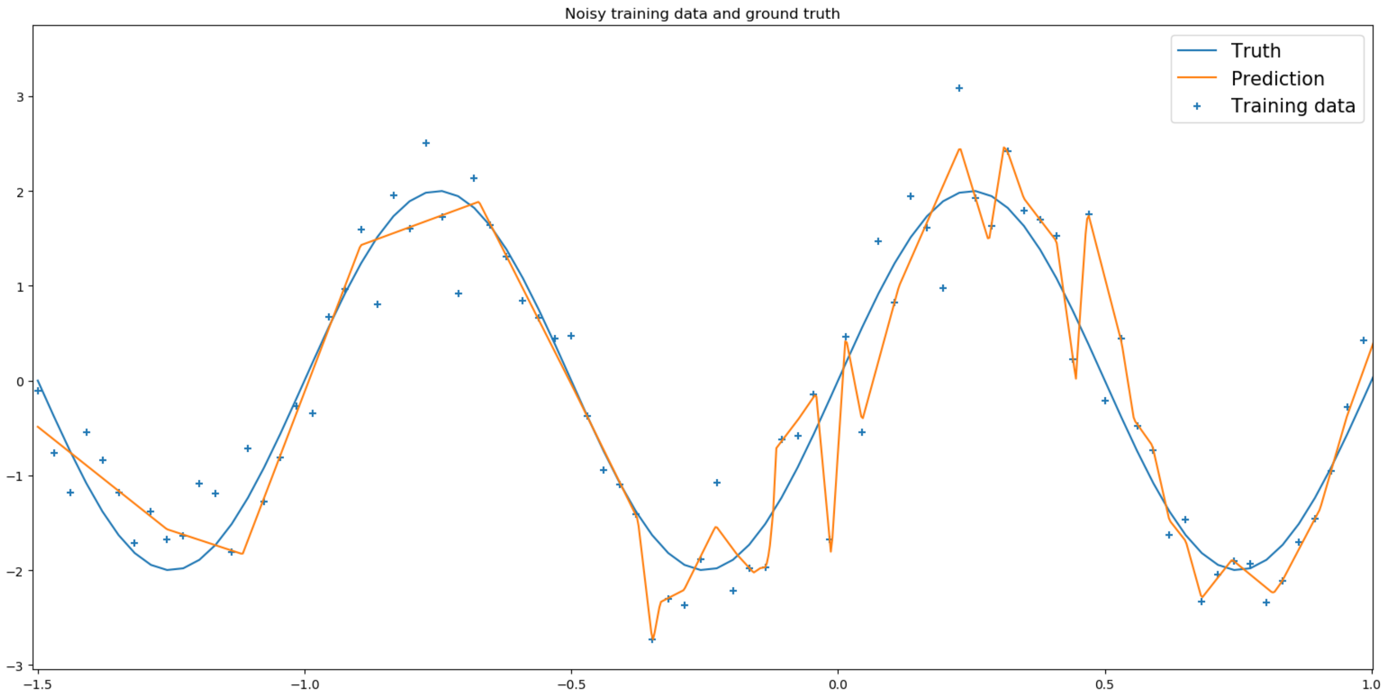
```
#set up data X,y

model = models.Sequential()
model.add(layers.Dense(16, input_dim=1, activation='relu'))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1))
model.compile(loss='mae', optimizer='adam')
model.fit(X, y, epochs=15000, batch_size=100)

#just some plotting
```

Problem: Overfitting

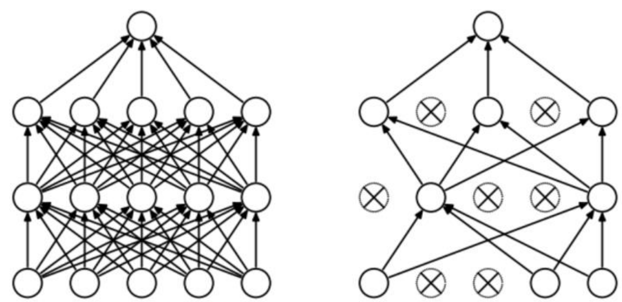
Network architecture: 1 – 16 – 32 – 16 – 1 (Relu)
10000 training epochs



Universal Approximation Theorem strikes back…

Solution: Overfitting

- Dropout
  - Randomly drop units from the neural network



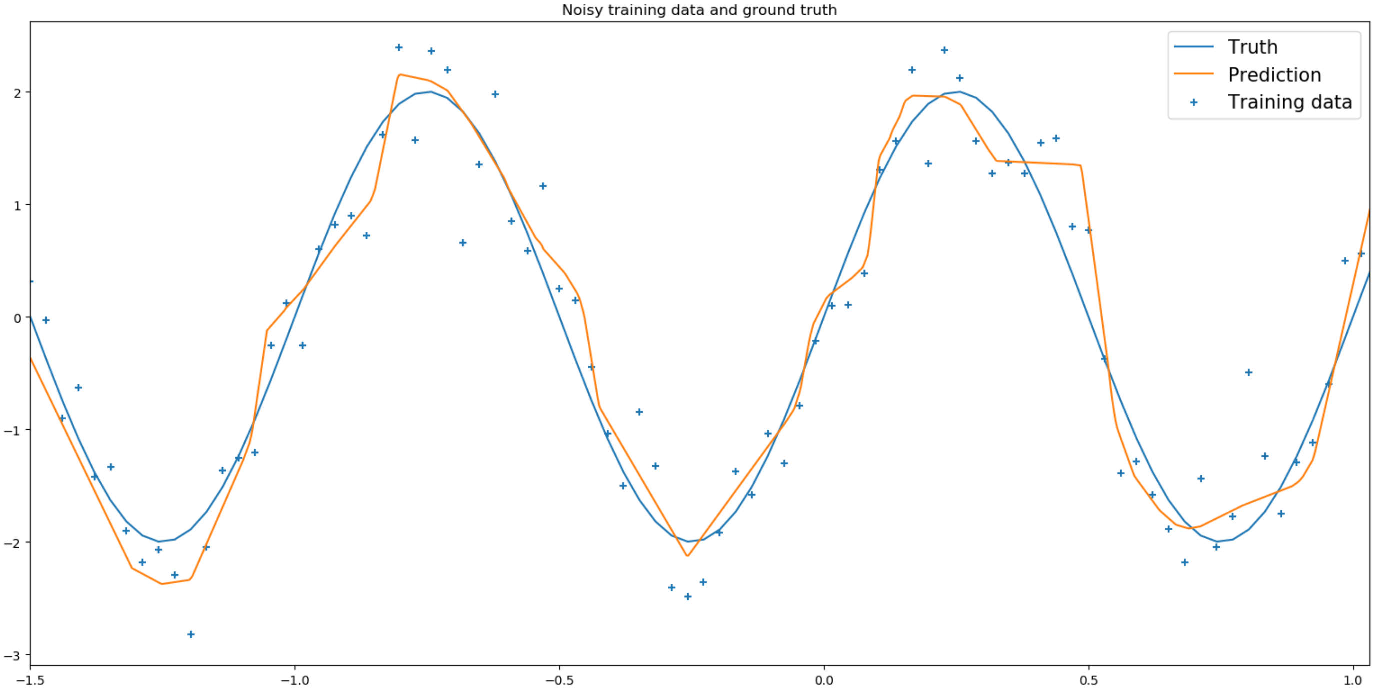
https://miro.medium.com/max/1044/1*iWQzxhVlvadk6VAJjsgXgg.png

- Kernel regularization
  - Adds regularizing term to loss function (e.g. L_2)

$$\text{loss} \to \text{loss} + \frac{\lambda}{2m} \sum_{l=1}^{L} ||w_l||^2$$

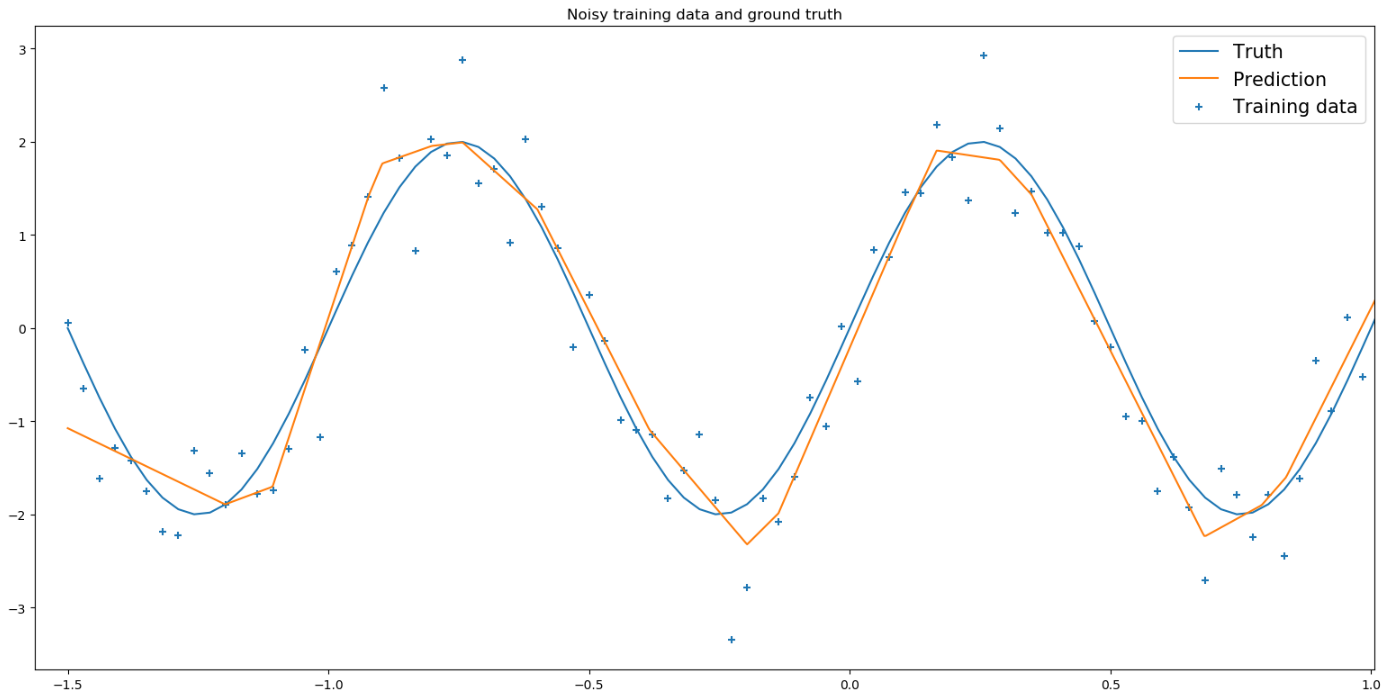
Solution: Dropout

Network architecture: 1 – 16 – 32 – 16 – 1 (Relu)
10000 training epochs
Dropout rate layer 3 = 0.2
Dropout rate layer 4 = 0.01

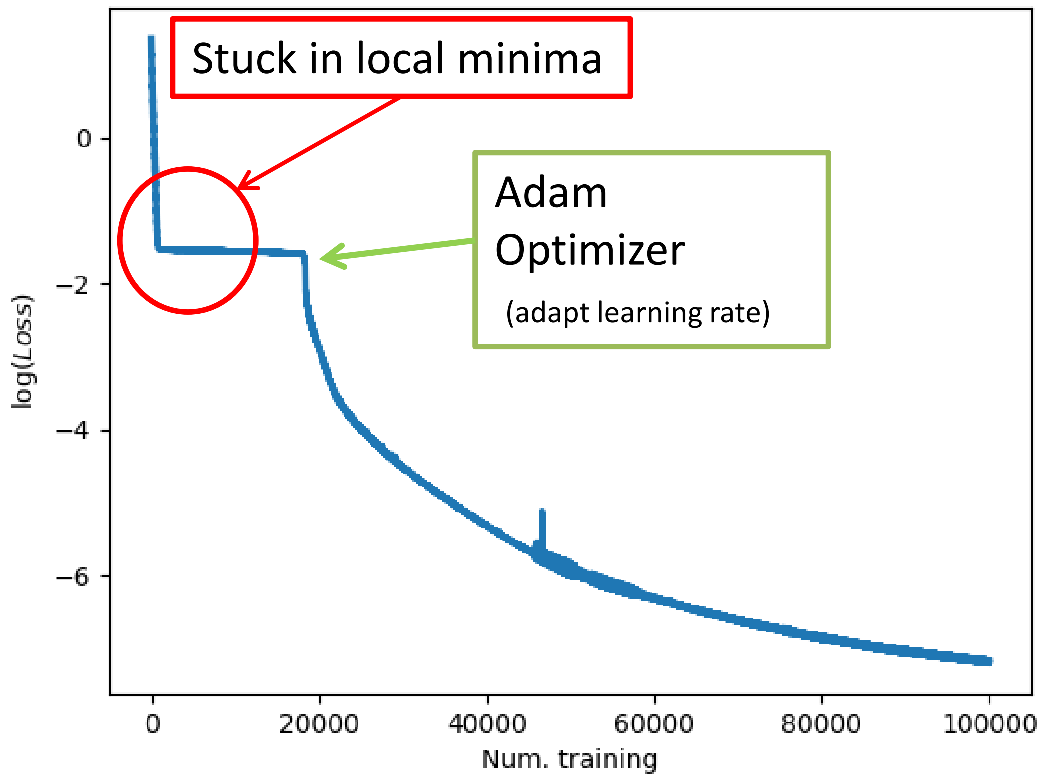


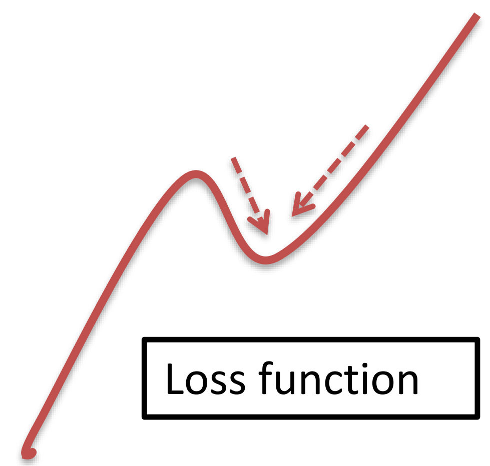
Solution: Regularization

Network architecture: 1 – 16 – 32 – 16 – 1 (Relu)
10000 training epochs
L2 regularization with $\lambda = 0.01$



Problem: Local minima



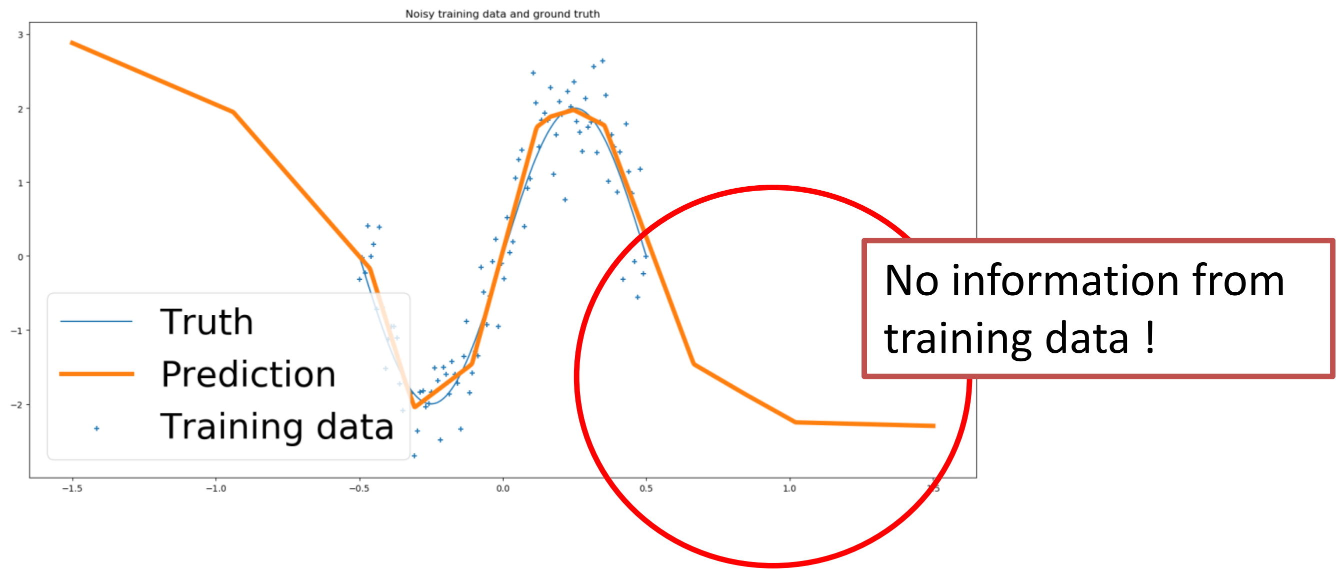


Summary Neural Networks

	• Simple implementation (even on GPUs) • Proven capacity • Scalability • …
	• Many hyperparameters: Network architecture, learning rate, number hidden neurons → <u>Trial and Error</u> • Overfitting, Estimating Uncertainties (Cross-Validation : Huge training / validation data set needed) • Black Box • And …

Fundamental Problem

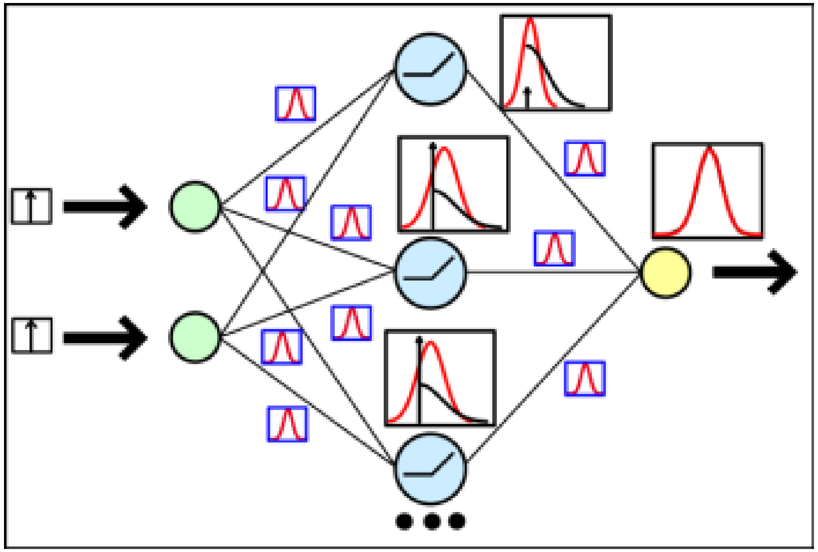
- **No** (good) information about **uncertainty** !



→We should go Bayesian....

Bayesian Neural Networks

x



$p(y|x, D)$

https://miro.medium.com/max/438/1*P48tfTLHDom0IbZkmG4RmQ.png

$$p(y|x, D) = \int \underbrace{p(y|x, w)}_{\text{Set to 1.}} \underbrace{p(w|D)}_{\text{Posterior over weights}} \mathrm{d}w$$

Set to 1.

Posterior over weights
→ **Intractable** because of nonlinearity

$D = \{x^{(i)}, y^{(i)}\}$ Training Dataset

w Weights

Approximation of the posterior

$$p(w|D)$$

- **Laplace Approximation**
  - Replace posterior by a Gaussian centered at the "true" posterior
- **MC Dropout**
  - Train Network with Dropout and sample from it using Dropout
- **Monte Carlo Methods**
- **Variational Inference**
  - Implemented in several software libraries (Pymc,…)

Variational Inference

- Bayes theorem $p(w|D) \propto p(D|w)p(w)$

  → Untractable → Approximate Posterior

- Find variational distribution $q(w|\theta)$ of known functional form (e.g. Gaussian $\theta = (\mu, \sigma)$, ...)

$$p(w|\theta) \overset{?}{\leftrightarrow} p(w|D)$$

↓

Minimize Kullback-Leibler divergence w.r.t to θ

Variational Inference

- The minimization problem can be written as:

$$\mathcal{F}(D,\theta) = \mathrm{KL}\left(q(w|\theta)||p(w)\right) - \mathrm{E}_{q(w|\theta)} \log(p(D|w))$$

$$\Big\downarrow \text{(...)}$$

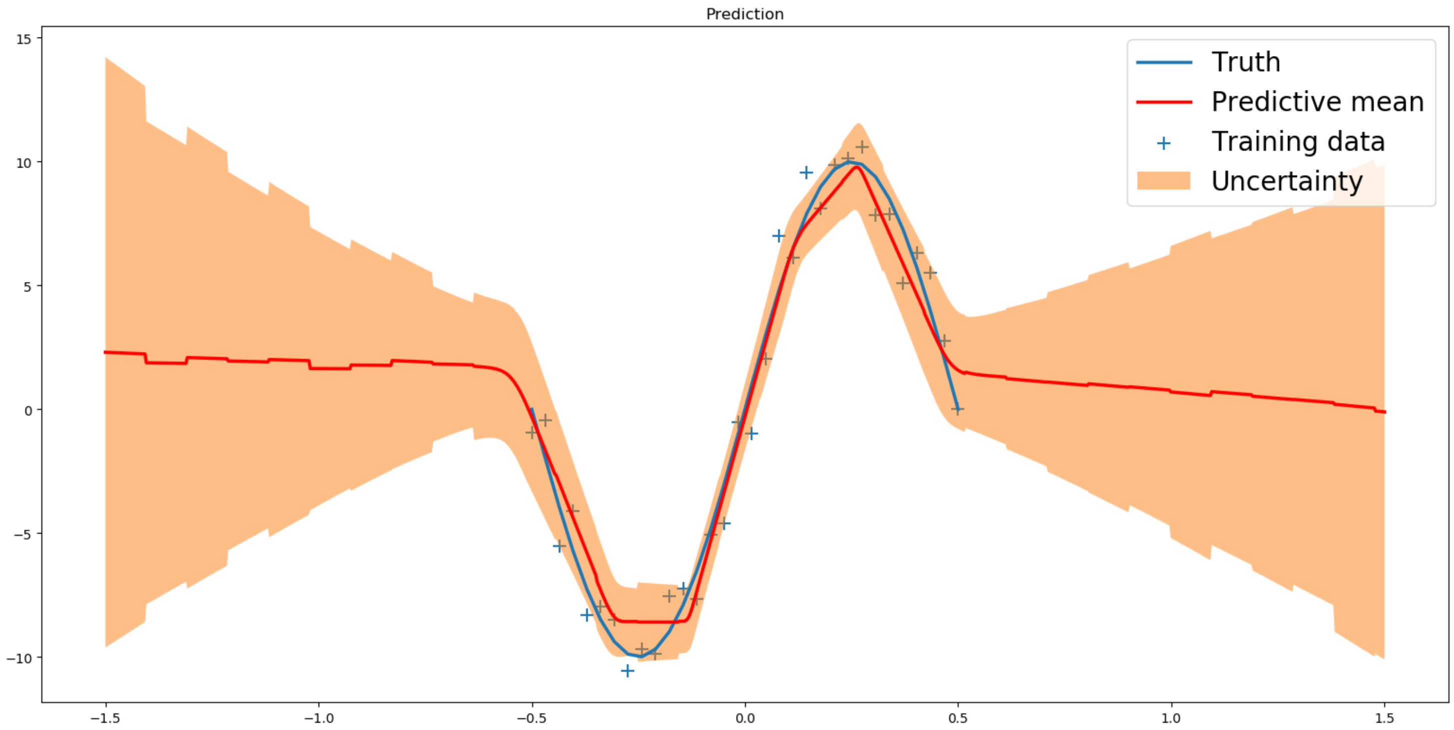
$$\mathcal{F}(D,\theta) \approx \underbrace{\frac{1}{N}\sum_{i=1}^{N}}_{\text{Draw N samples}} \Big[\underbrace{\log(q(w^{(i)}|\theta) - \log(p(w^{(i)}))}_{\text{Complexity cost}} - \underbrace{\log(p(D|w^{(i)}))}_{\text{Likelihood cost}} \Big]$$

Variational Inference: Network Training

Forward-pass:	-Draw sample from variational posterior -Evaluate cost function $\mathcal{F}$ with sample
Backward-pass:	-Calculate gradients of (μ, σ) <u>-BUT:</u> stochastic sampling $\leftrightarrow$ gradient ? → re-parameterization with deterministic function (where gradient can be defined)

Bayesian Neural Network: Example

Network structure: 1-20-20-1 (Relu)
5000 training epochs
1000 samples for evaluation



Bayesian Neural Network: Uncertainty

- **About what is our network uncertain?**
  - Uncertainty of weights?
  - Uncertainty about observation caused by noisy data set?

→ The example can only cover uncertainty of weights (we set $p(y|x, w) = 1$)

Summary Bayesian Neural Networks

	• Calculate error associated with predictions (BUT: future research required, efficient implementation) • No validation data needed • Avoids overfitting • Less weights and training data needed
	• Difficult to scale up • Approximations to Bayesian approach (e.g. uncorrelated weights)

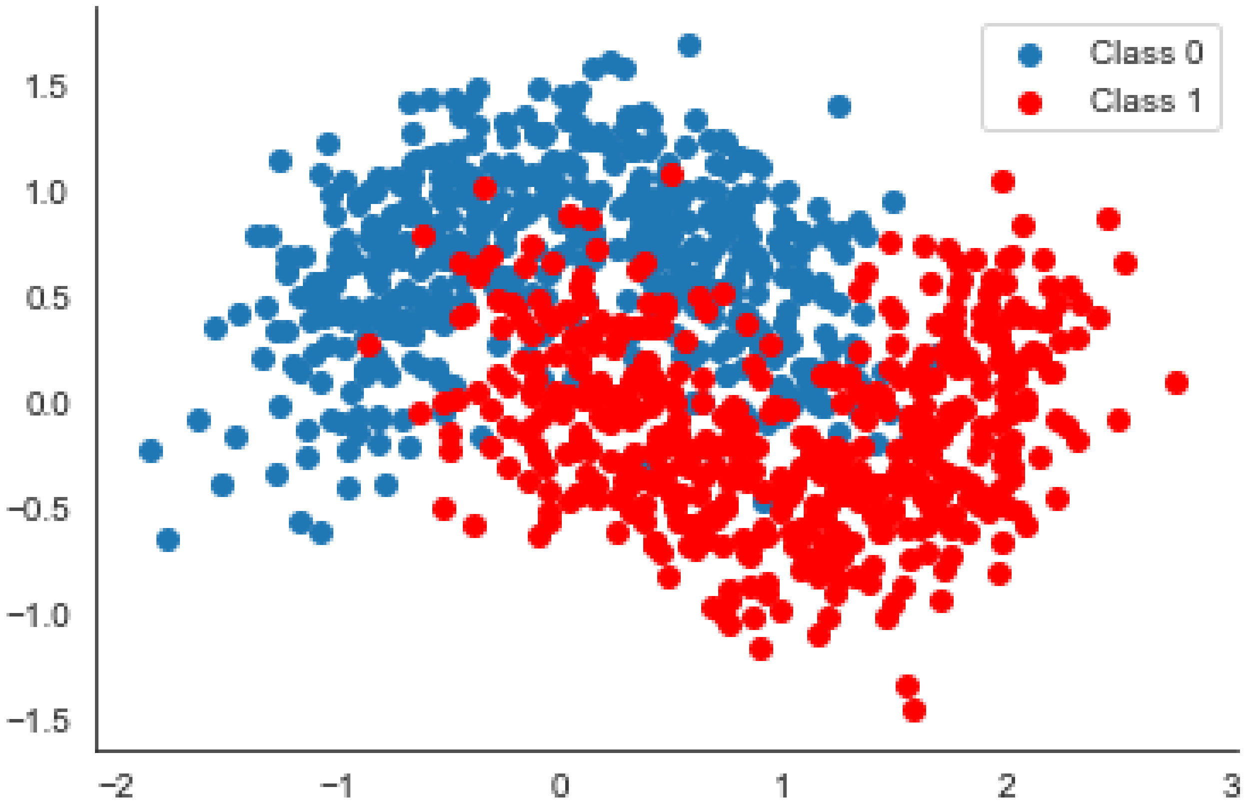
Hierarchical Bayesian Neural Networks

Example following:
https://twiecki.io/blog/2018/08/13/hierarchical_bayesian_neural_network/

(all figures taken from there)

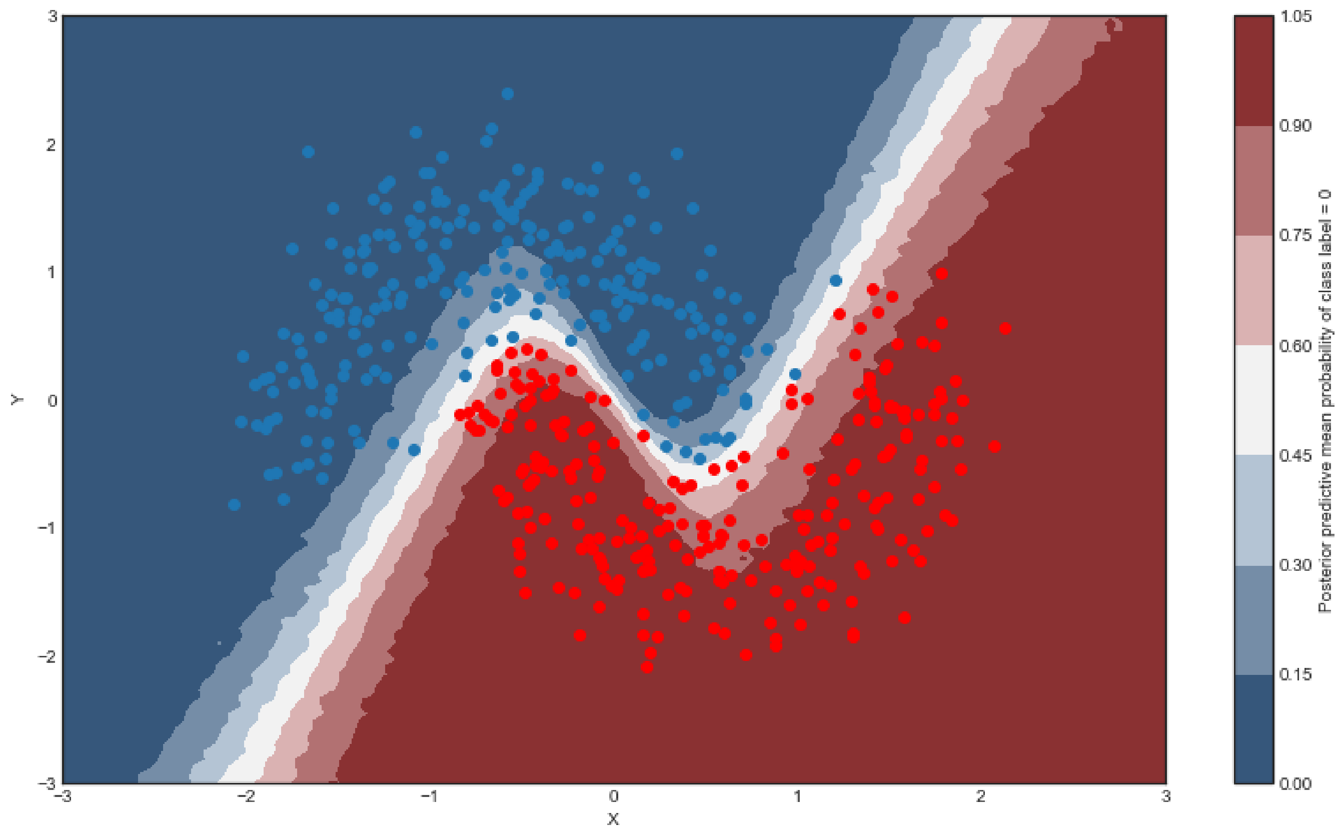
Hierarchical Bayesian Neural Networks

- Classification Problem



Hierarchical Bayesian Neural Networks

- Train "normal Bayesian NN"



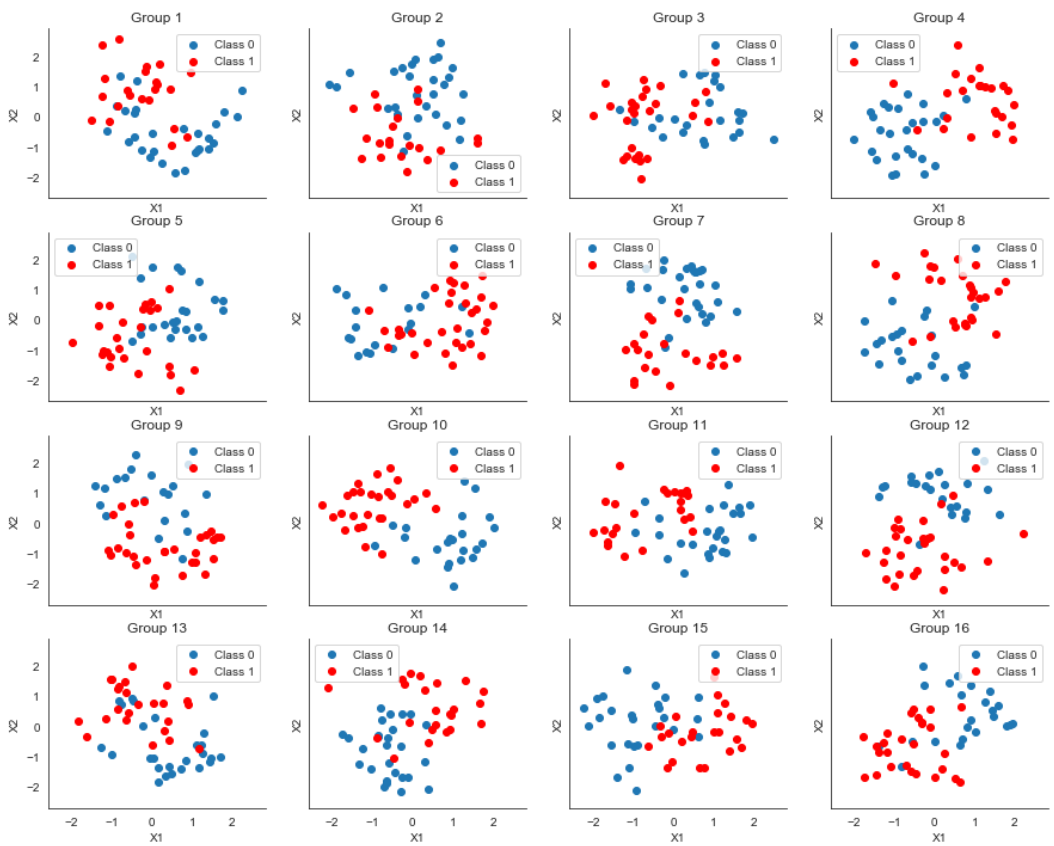
"Z-Shape"

Hierarchical Bayesian Neural Networks

Find the limits of "normal" Bayesian Neural Network:

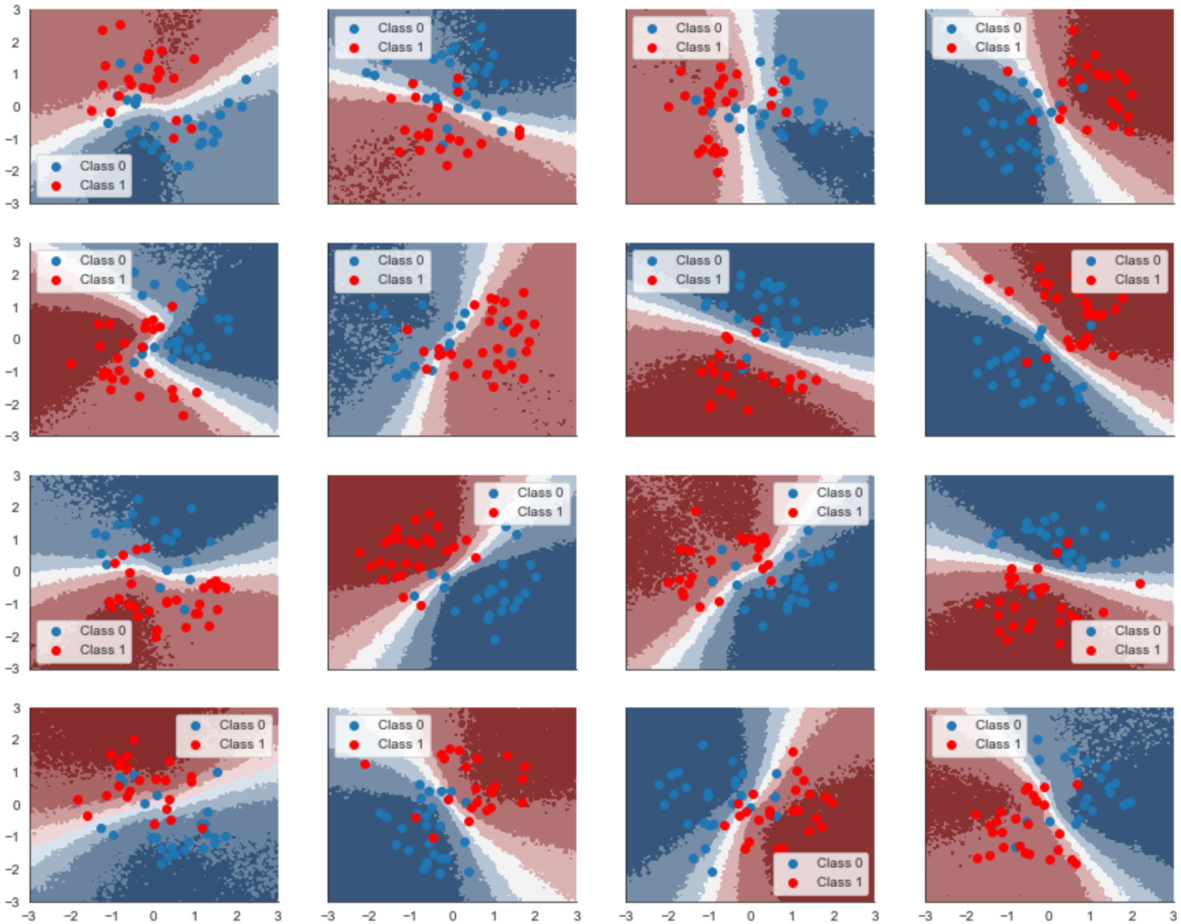
-Generate 16 different groups by rotating the data

-only a few training points per group



Hierarchical Bayesian Neural Networks

- Train one normal BNN for every group:



- "Z-Shape" not detected
- We should take advantage of the <u>similarities between the groups</u>

Hierarchical Bayesian Neural Networks

From Moritz' talk ....

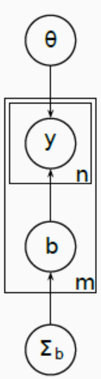
A simple example

Simple Hierarchical

$\theta \sim$ shared parameters
$b_i \sim$ cluster-specific param.
$\Sigma_b \sim$ distr. over b_i
$y_{ij} \sim$ observables

Setup

$\theta = (\mu, \sigma_b^2, \sigma_y^2)$ shared
$b_i \sim \mathcal{N}(0, \sigma_b^2)$ deviations
$\mu_i = \mu + b_i$ specific mean
$y_{ij} \sim \mathcal{N}(\mu_i, \sigma_y^2)$

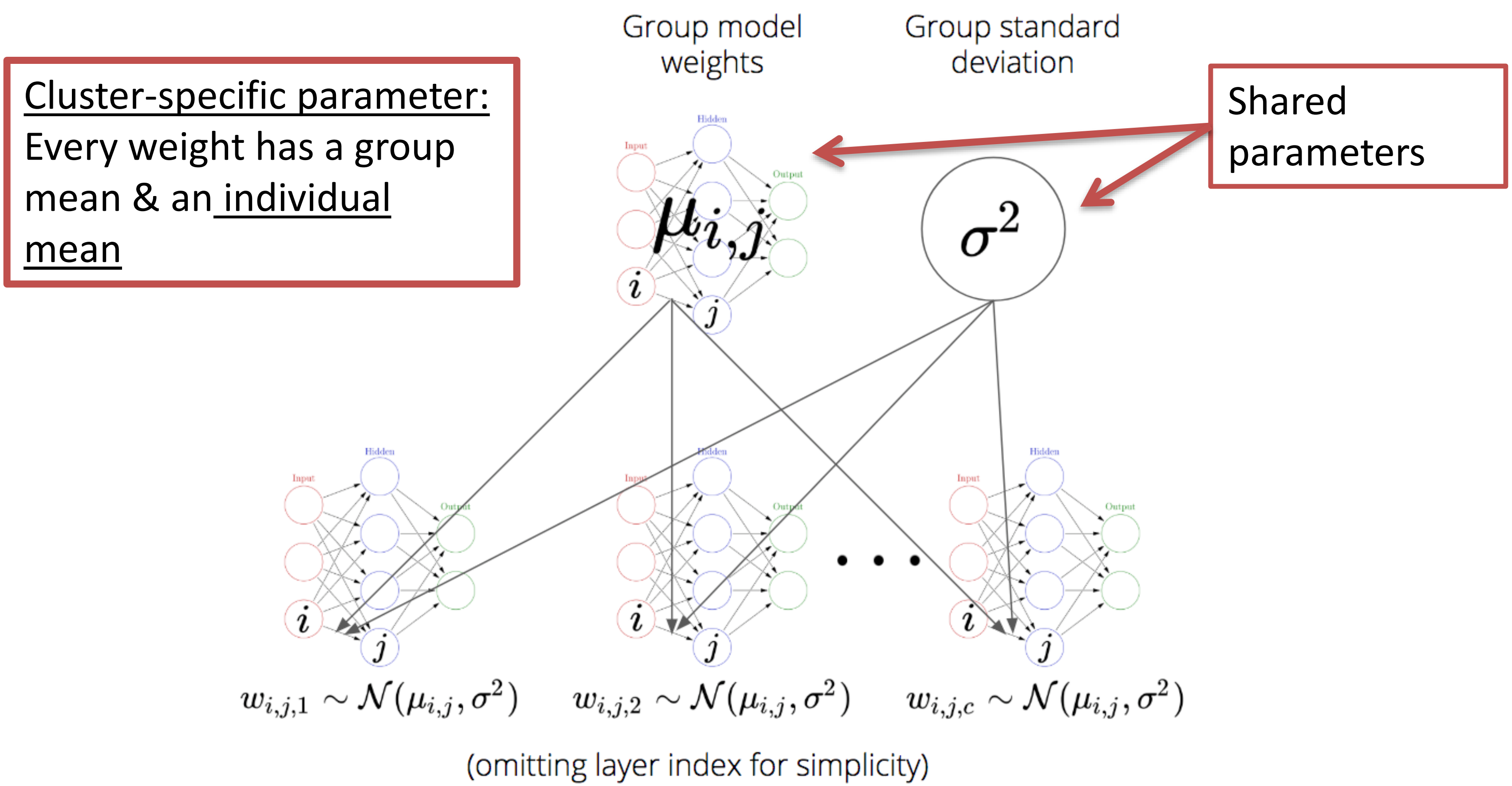


128 x 96 mm

6

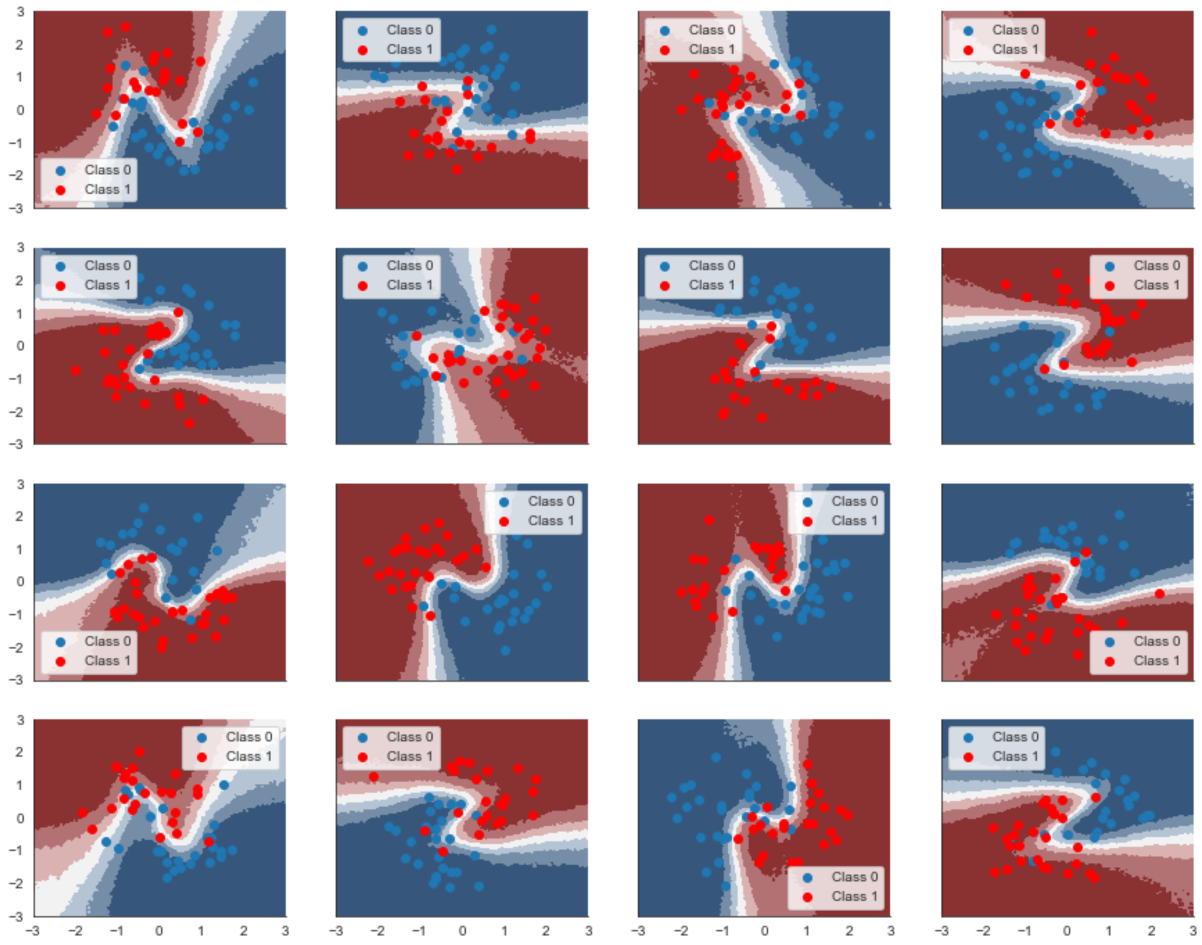
Hierarchical Bayesian Neural Networks

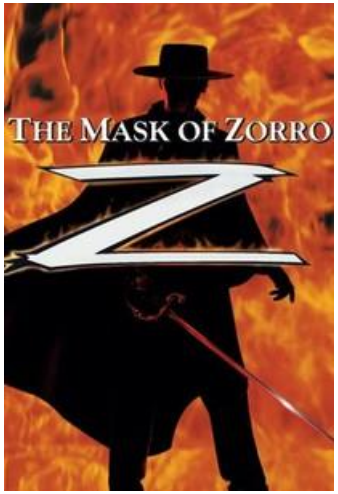
- Train a hierarchical model:



Hierarchical Bayesian Neural Networks

- Train a hierarchical model:

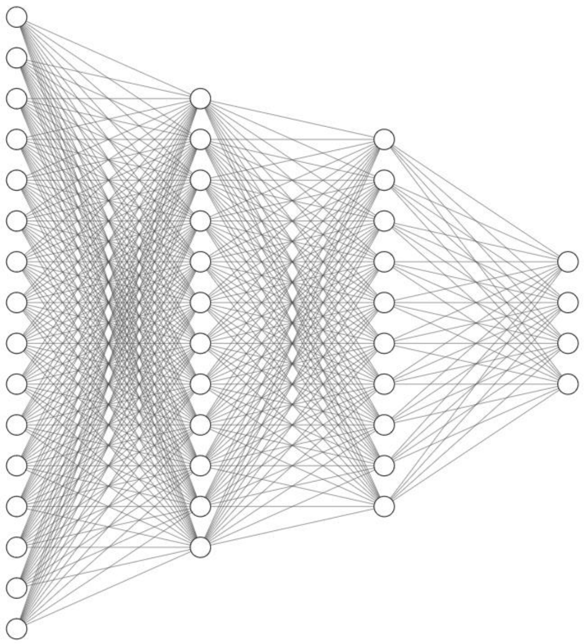




https://resizing.flixster.com/JITvzaGqtRxPUlUFXTrZo2wQ9ZI=/206x305/v1.bTsxMTE2OTU5NDtqOzE4MzU0OzEyMDA7ODAwOzEyMDA

Automatic Relevance Determination

- Imagine dataset with many features, but not all are important



→Bad performance, because weights of irrelevant inputs are not set to 0
→Determine relevant inputs

Automatic Relevance Determination

- Set prior with hyperparameter α on weights:

$$P(w|\alpha_i) \propto \mathcal{N}(0, \alpha_i)$$

- Posterior over weights becomes:

$$P(w, \alpha|D) = \underbrace{P(w|D, \alpha)}\,\underbrace{P(\alpha|D)}$$

Assumption Bayes

$$P(w, \alpha|D) \propto \overbrace{\mathcal{N}(\mu, \Sigma)}\,\overbrace{P(D|\alpha)P(\alpha)}$$

$\boxed{\text{Controlled by } \alpha}$

Automatic Relevance Determination

$$P(w, \alpha | D) \propto \mathcal{N}(\mu, \Sigma) P(D|\alpha) P(\alpha)$$

- Maximize likelihood $P(D|\alpha)$ (assume uniform hyperprior $P(\alpha)$):

$$\boxed{P(D|\alpha) = \int P(D|w) P(w|\alpha) \mathrm{d}w}$$

$\Big\downarrow$ (…)

$$\boxed{\frac{\mathrm{d}}{\mathrm{d}\alpha_i} \ln(P(D|\alpha)) = 0} \xrightarrow{(\dots)} \boxed{\alpha_i = \frac{1 - \alpha_i \Sigma_{ii}}{\mu_i^2}}$$

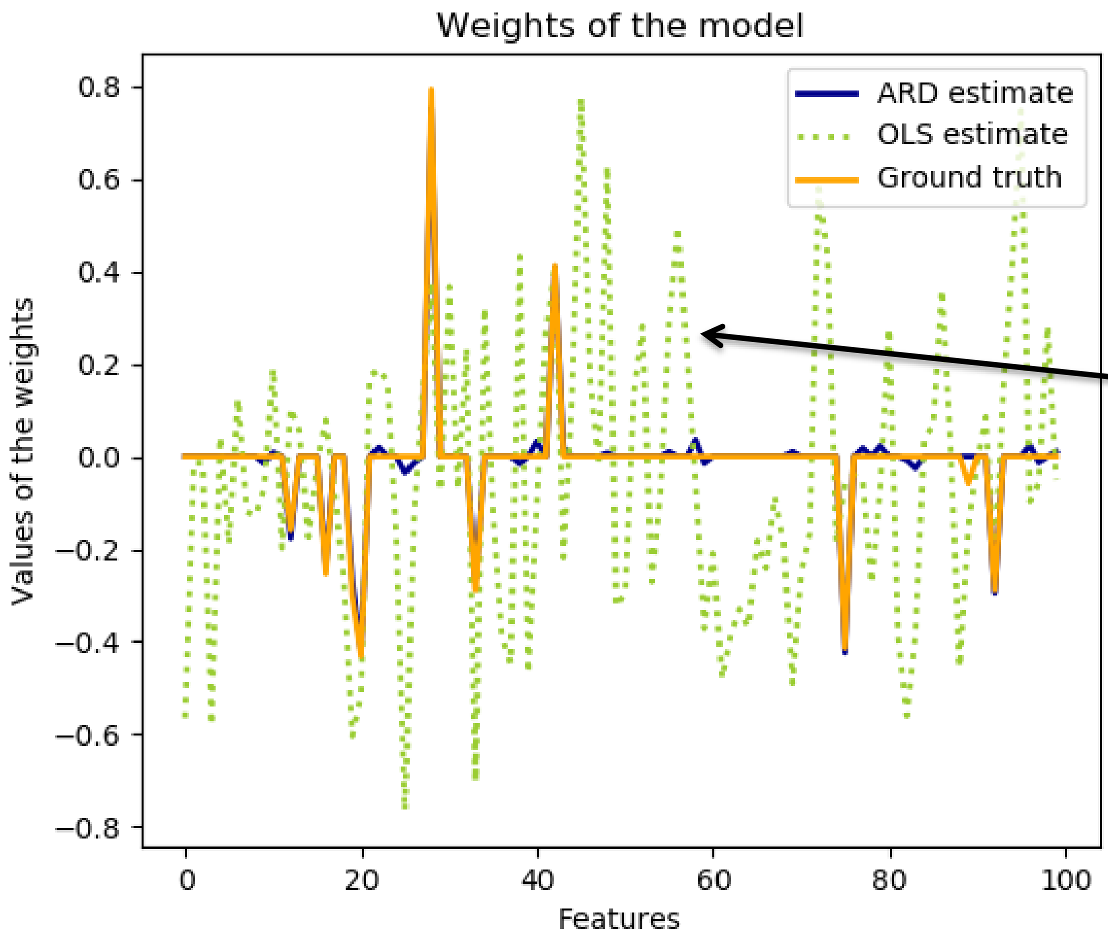
→Implicit equation
(Solved by Iteration)

Automatic Relevance Determination

- Some α will tend to ∞ → weight prior is $\mathcal{N}(0, 0)$ distributed → weight is 0

- ARD can be applied to a huge amount of Models (Regression, Neural Networks,...)
- Let's have a look at an example...

Automatic Relevance Determination



Model weights of "good" model with useless features

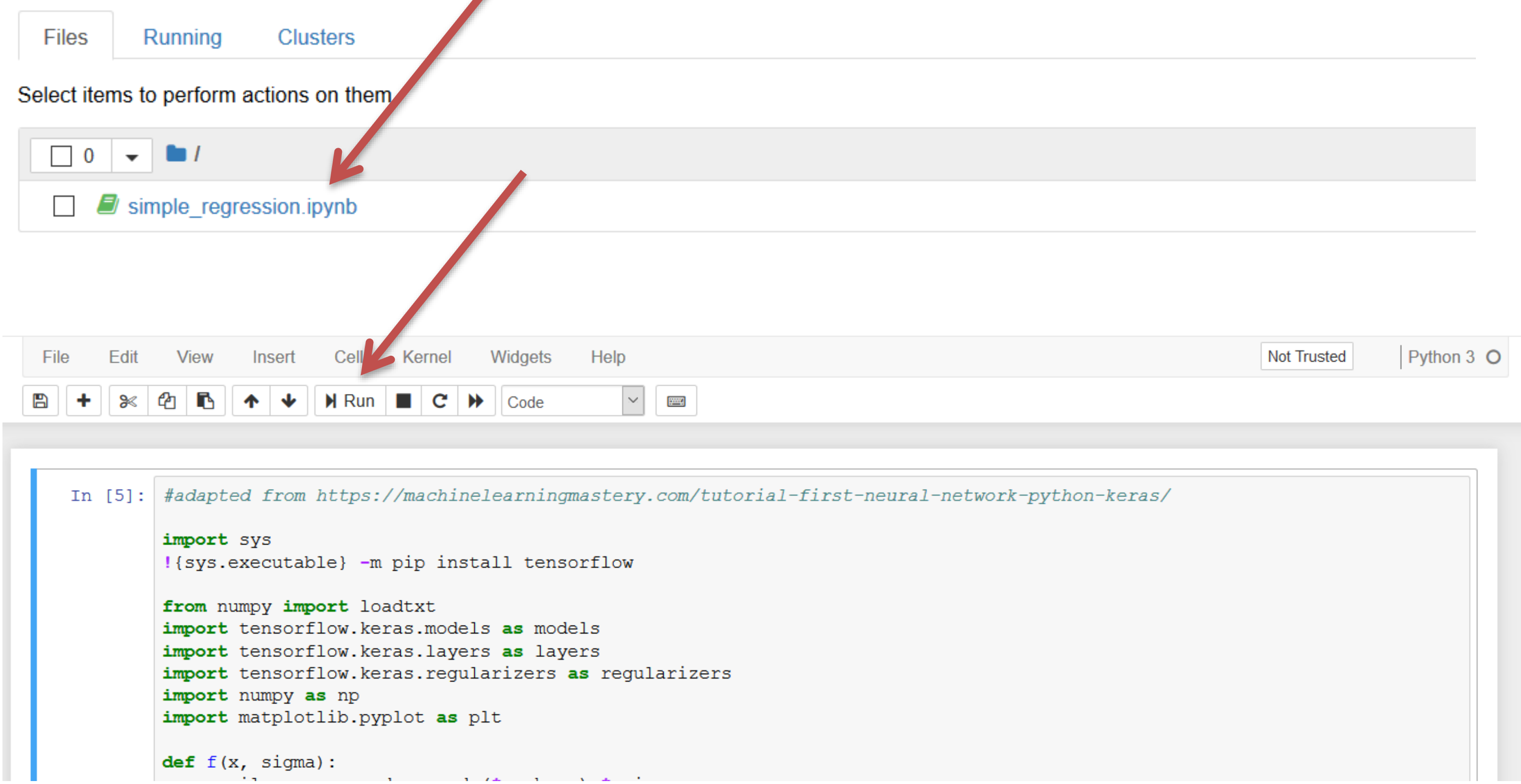
https://scikit-learn.org/stable/_images/sphx_glr_plot_ard_001.png

Take-Home Message

Problem / Task	Solution
Nonlinear Regression, Classification	Neural Network
Uncertainty Estimation	Bayesian Neural Network
Learn from different groups with similarities & Transfer Learning	Hierarchical Bayesian Neural Network
Dataset with irrelevant features	Automatic Relevance Determination

Regression example

- https://mybinder.org/v2/gh/blaschma/regression_example/master



References

- Bishop, Christopher M. *Pattern recognition and machine learning*. springer, 2006.
- Neal, Radford M. *Bayesian learning for neural networks*. Vol. 118. Springer Science & Business Media, 2012.
- Fletcher, Tristan. "Relevance vector machines explained." *University College London: London, UK* (2010).
- https://twiecki.io/blog/2018/08/13/hierarchical_bayesian_neural_network/
- http://krasserm.github.io/2019/03/14/bayesian-neural-networks/